작성자(학번 이름) : 2019180052 한창우

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **주차** | 7주차 | **기간** | 2024.8. 5 ~ 8. 11 | **지도교수** |  |
| 이번주 한일 요약 | Windows Via C/C++  챕터 8 “커널 모드 스레드 동기화” | | | | |

<상세 수행내용>

**커널 오브젝트를 이용한 스레드 동기화**

커널 오브젝트를 이용한 동기화는 유저 모드 스레드 동기화 메커니즘 보다는 성능이 안좋지만 활용성과 안정성은 더 뛰어나다. 대부분의 커널 오브젝트는 동기화를 위해 사용될 수 있다.

커널 오브젝트는 **“시그널”** 과 **“논시그널”** 상태를 가진다.

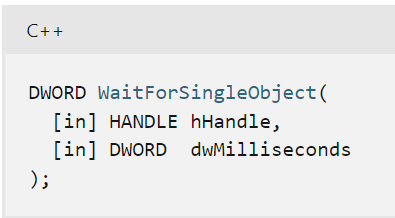
프로세스는 생성되면 처음에 **“FALSE(논시그널)”** 값을 가진다. 그리고 프로세스가 종료되면 운영체제에 의해 “**TRUE(시그널)”** 값을 가진다. 이는 스레드 커널 오브젝트 또한 이와 동일한 규칙을 따른다. 즉, 스레드 커널은 논시그널 상태로 생성되며, 스레드가 종료되면 운영체제가 자동으로 오브젝트의 상태를 시그널 상태로 변경한다.

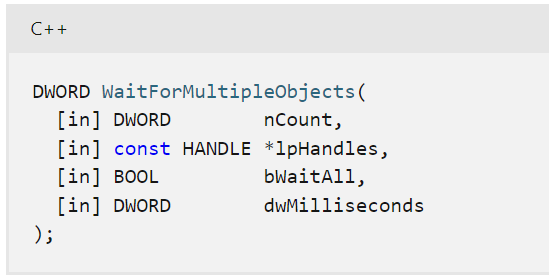
다음에는 시그널 상태나 논시그널 상태가 될 수 있는 커널 오브젝트들이다.

* 프로세스
* 스레드
* 잡
* 파일과 콘솔에 대한 표준 입력/출력/에러 스트림
* 이벤트
* 대기 타이머
* 세마포어
* 뮤텍스

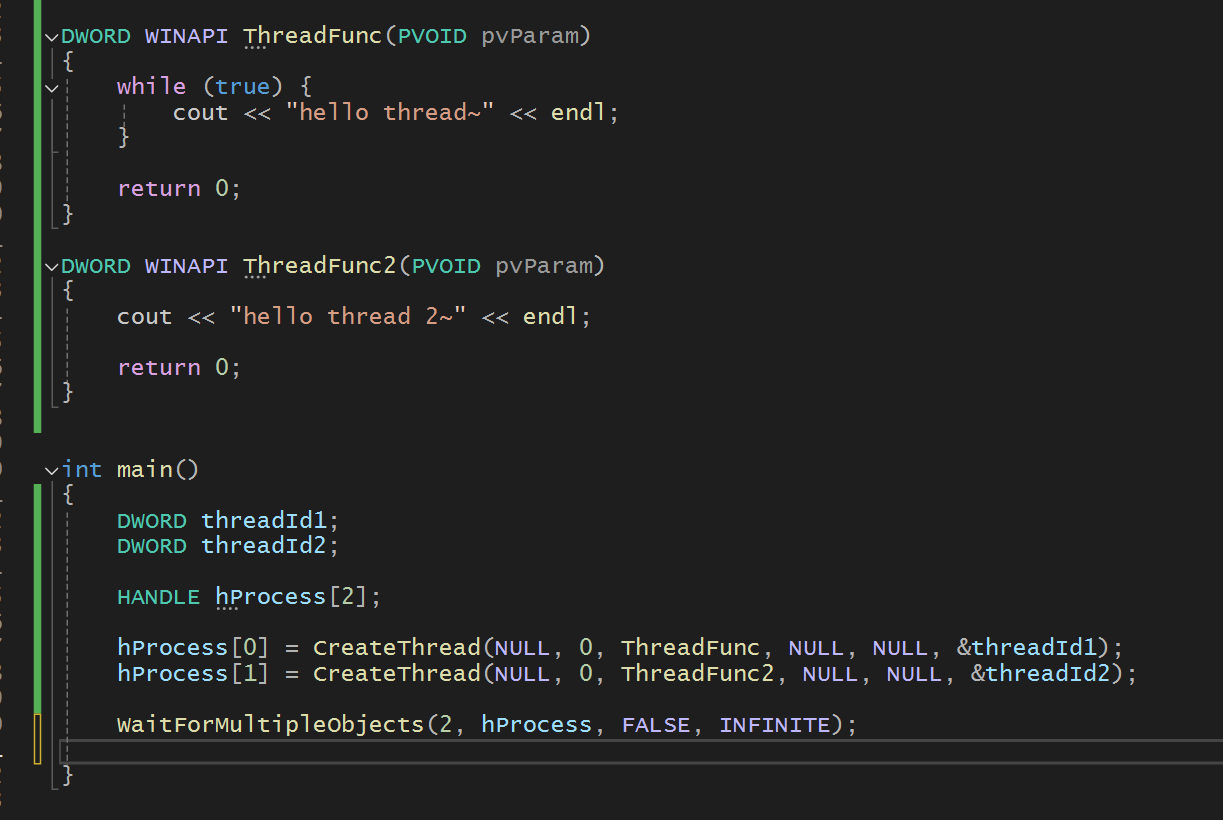
**대기 함수들**

대기 함수를 호출하면 인자로 전달한 커널 오브젝트가 시그널 상태가 될 때까지 이 함수를 호출한 스레드를 대기 상태로 유지한다.





위 함수들은 이전 일지에서 사용해봐서 자세한 설명은 생략하고 새로 알게된 사실만 쓰겠다.



위 코드에서 WaitForMultipleObjects 함수의 3번째 매개변수로 FALSE를 전달했다. 이 뜻은 시그널 상태가 되기를 기다리는 커널오브젝트들 중에서 하나라도 시그널이 되면 블로킹을 빠져나오게 된다. ThreadFunc를 실행하는 스레드는 종료가 되지않았음에도 불구하고 ThreadFunc2를 실행하던 스레드가 종료되어 WaitForMultipleObjects를 실행하던 스레드는 블로킹에서 빠져나오게 되고 프로그램이 종료되었다. 이는 ThreadFunc에서 사용하던 리소스가 있었다면 반환하지 못한다는 뜻이다.

그렇다면 몇번째 커널오브젝트가 시그널 상태가 되어서 반환된건지 알 방법은 무엇인가?



WaitForMultipleObjects가 반환한 값을 dw에 저장한다. 그리고 이 dw에 WAIT\_OBJECT\_0 값을 더하여서 몇번째 인덱스인지 알 수 있다. 하지만 해당 인덱스에 있는 커널 오브젝트는 제거된 새로운 배열을 만들어야 WaitForMultipleObjects함수를 재호출할 때 매개변수로 쓸 수 있다.

그렇지 않고 시그널 상태의 오브젝트가 포함된 배열을 그대로 WaitForMultipleObjects 함수의 매개변수로 전달하면, WaitForMultipleObjects 함수는 바로 WAIT\_OBJECT\_0 값을 반환하여 정상적으로 수행되지 않을 것이다.

**성공적인 대기의 부가적인 영향**

성공적인 호출을 통해 오브젝트의 상태가 변경되는 것을 일컬어 **"성공적인 대기의 부가적인 영향"** 이라고 한다. 예를 들어 어떤 스레드가 “**자동 리셋 이벤트 커널 오브젝트**” 핸들을 매개변수로 대기 함수를 호출하는 경우, 이 오브젝트가 시그널 상태가 되면 WAIT\_OBJECT\_0을 반환할 것이다. 또한 함수가 반환되기 직전에 **“성공적인 대기의 부가적인 영향”**으로 인해 이벤트 커널 오브젝트는 논시그널 상태로 변경될 것이다. 이렇게 부가적인 영향을 가지는 오브젝트들이 있고 그렇지 않은 오브젝트들이 있다.

WaitForMultipleObjects 함수가 커널 오브젝트들의 상태를 확인할 때는 다른 스레드가 그 상태를 변경할 수 없다. 만약 두 스레드가 각각 다른 커널 오브젝트를 시그널 상태에서 논시그널 상태로 변경하면, 이로 인해 교착 상태(Deadlock)가 발생할 수 있는데, 이러한 교착 상태는 서로 다른 스레드가 서로의 이벤트에 의존하면서 발생하며, 결과적으로 스레드들이 완전히 정지하게 된다. 하지만 WaitForMultipleObjects는 모든 작업을 원자적으로 수행하기 때문에 유용하다.

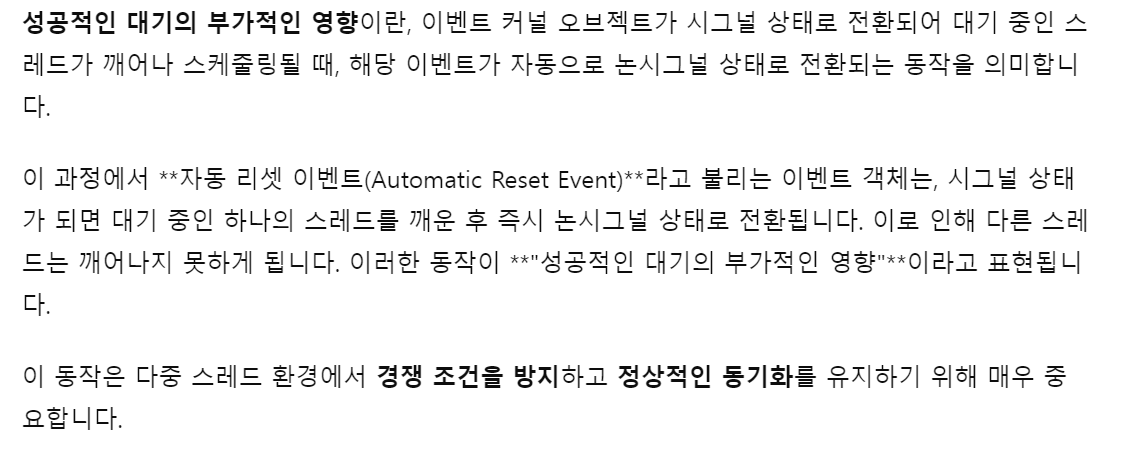
**이벤트 커널 오브젝트**

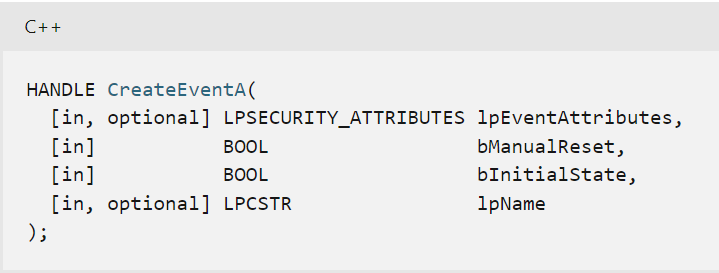
이벤트 커널 오브젝트는 단순한 구조를 가지고 있다. **“사용 카운트”**, **“자동 리셋 이벤트 인지 수동 리셋 이벤트인지 알려주는 bool값”**, 그리고 마지막으로 **“시그널, 논시그널 값”** 을 가지고 있다.

여기서 자동, 수동 리셋 이벤트의 설명은 다음과 같다.

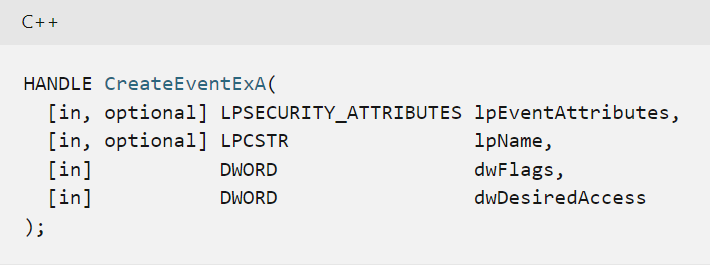
**자동 리셋 이벤트** : 대기 중인 스레드들 중에 하나의 스레드만이 스케줄 가능 상태가 되는 것

**수동 리셋 이벤트** : 대기 중인 모든 스레드들이 동시에 스케줄 가능 상태가 되는 것

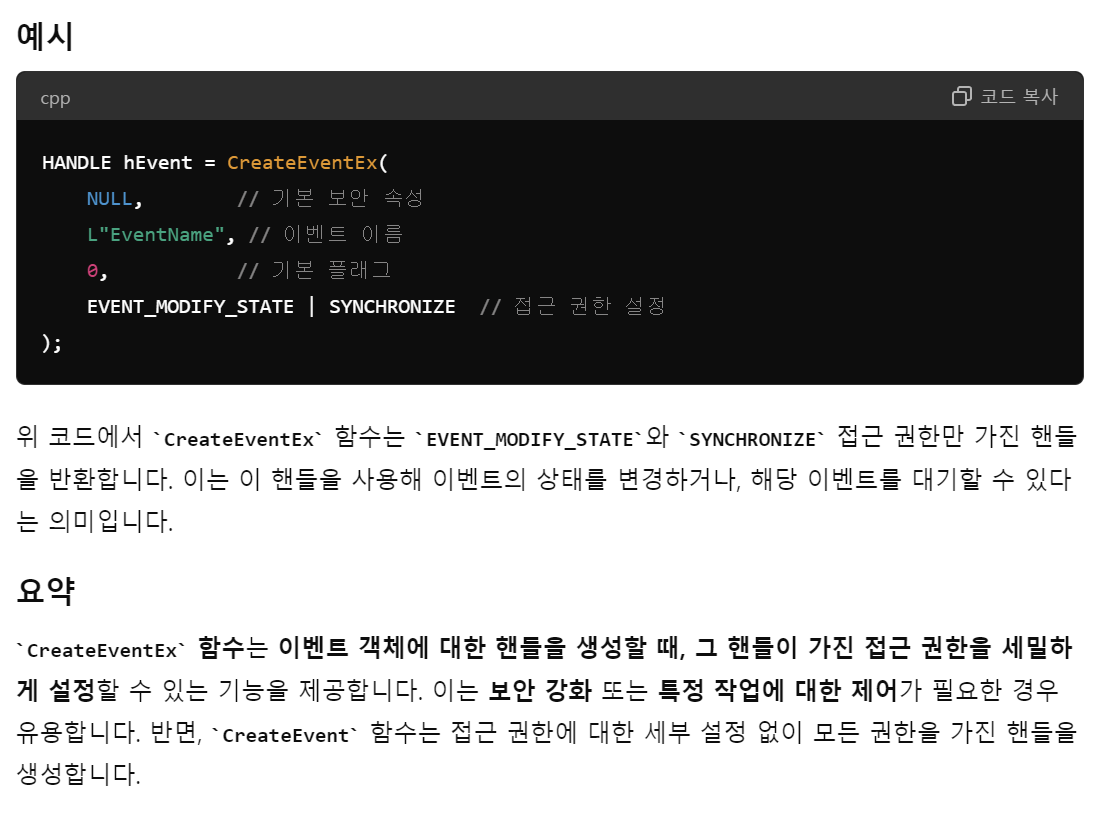
(gpt 설명 참조)

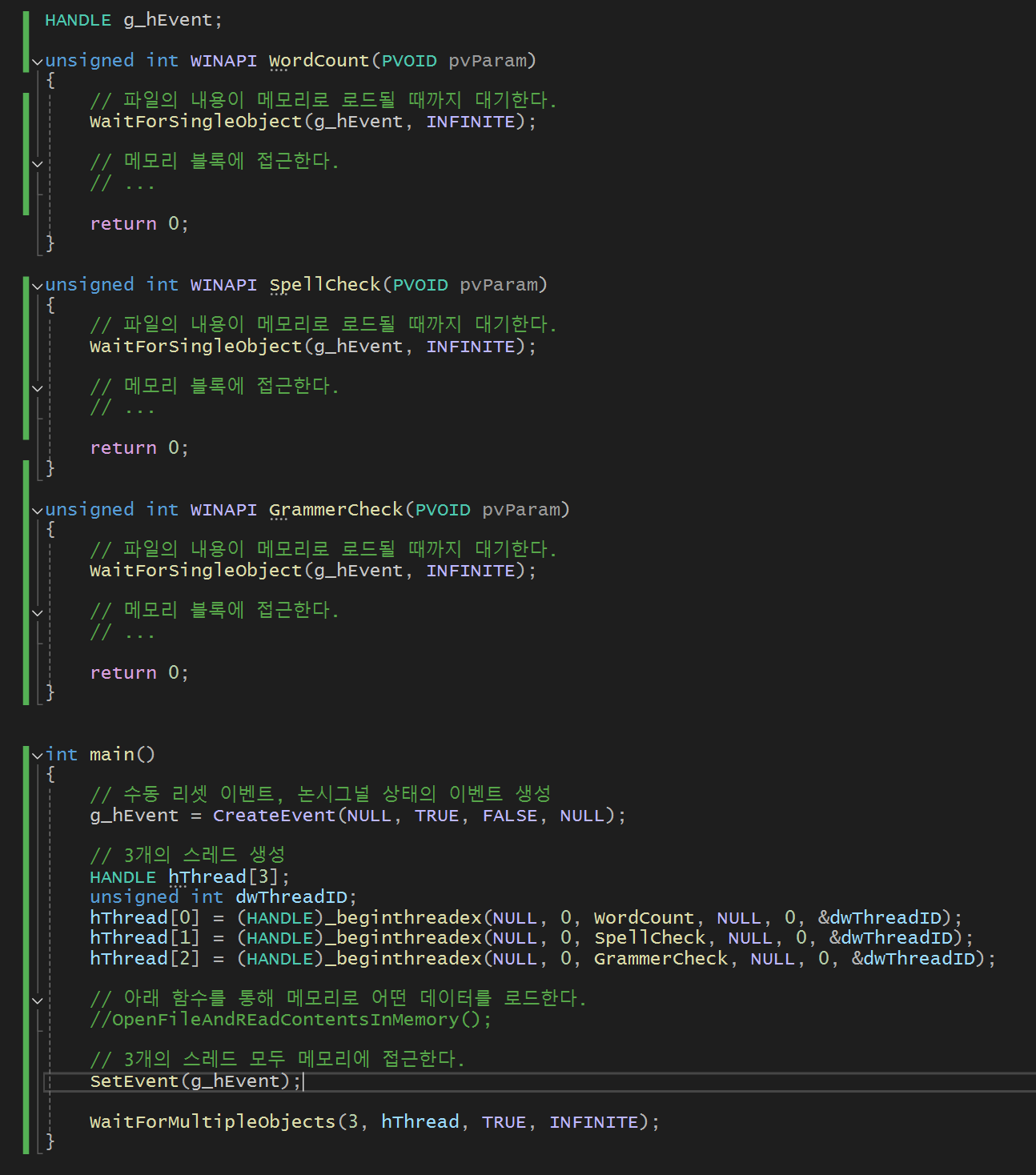


* bManalReset 매개변수는 자동 또는 수동 리셋 이벤트를 만들건지에 대한 여부
* bInitialState 매개변수는 이벤트의 초기 상태를 시그널 또는 논시그널로 설정할건지에 대한 여부



dwFlags 값으로 위 CreateEvent함수와 동일한 것들을 설정할 수 있고, dwDesiredAcess 값으로 해당 이벤트 커널 오브젝트에 접근하는 권한을 설정하게 된다.

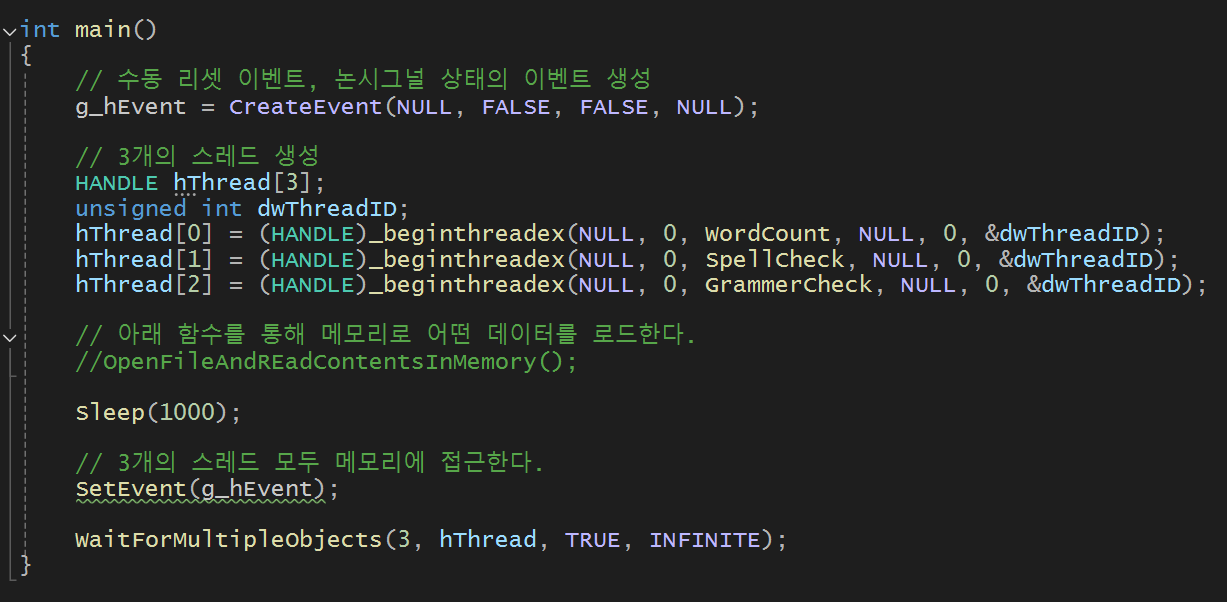
**CreateEvent 와 CreateEventEx함수의 차이점을 gpt에 검색한 결과이다.**

**이벤트 객체 수동 리셋**

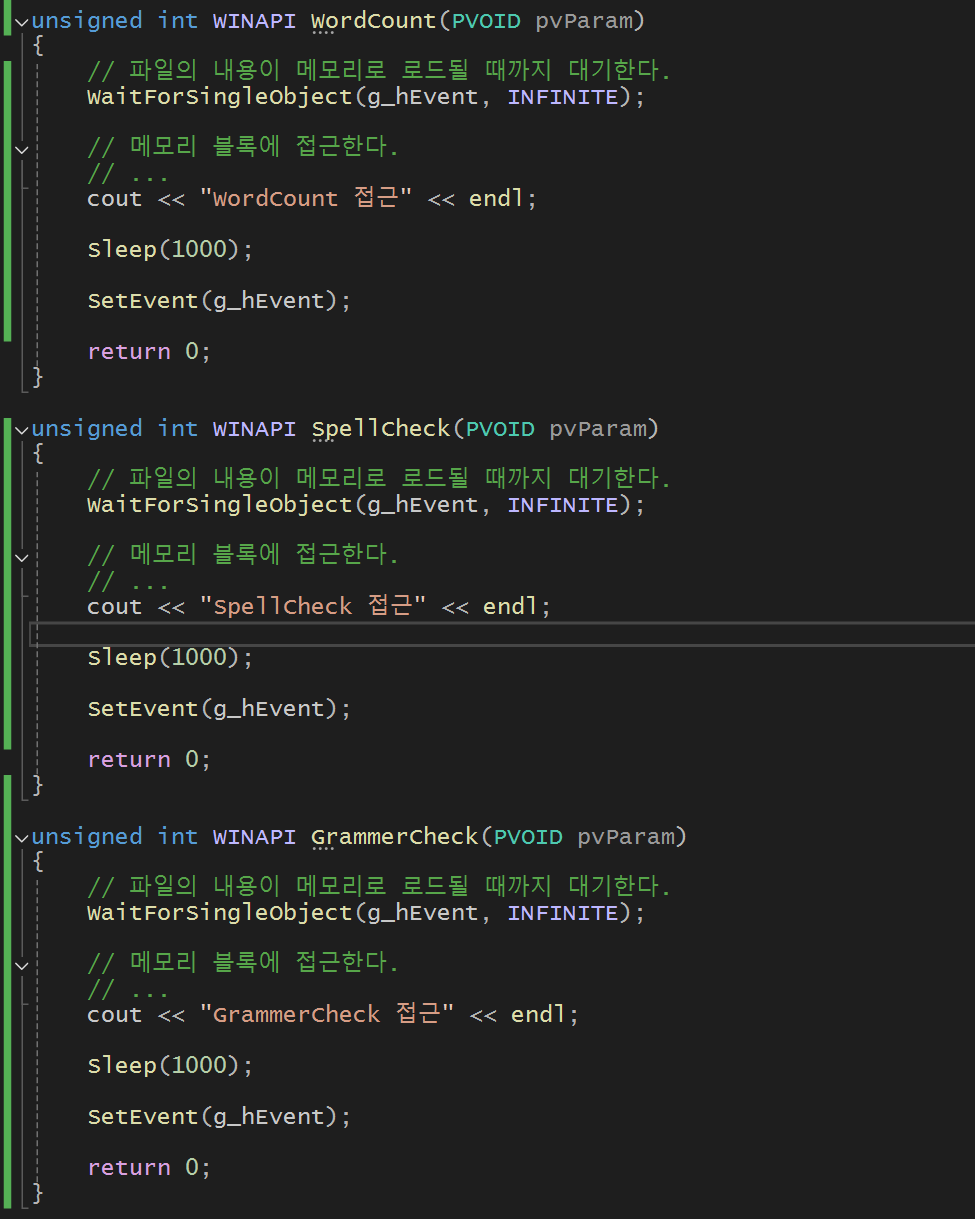
위 코드는 다음과 같은 상황이다.

수동 리셋 이벤트 객체를 생성하였고, 메인 스레드에서 어떤 파일을 메모리로 로드한다.

각 스레드에서는 메인에서 파일을 메모리로 로드하기 전까지 WaitForSingleObjet함수로 대기한다. 그리고 메인에서 로드가 끝나면 SetEvent로 이벤트를 시그널 상태로 만들어주면, 3개의 스레드는 해당 메모리에 접근하여 작업을 하게 되는 것이다.

**이벤트 객체를 자동 리셋으로 수정**

이번에는 자동 리셋 이벤트를 생성하여서 스레드들이 한번에 공유 데이터에 접근하는 것을 막고, 하나의 한 스레드만 배타적을 접근할 수 있도록 수정하였다.



메인 스레드에서 SetEvent로 시그널 상태로 바꾸면 대기 중이던 하나의 스레드만 접근하게 된다.

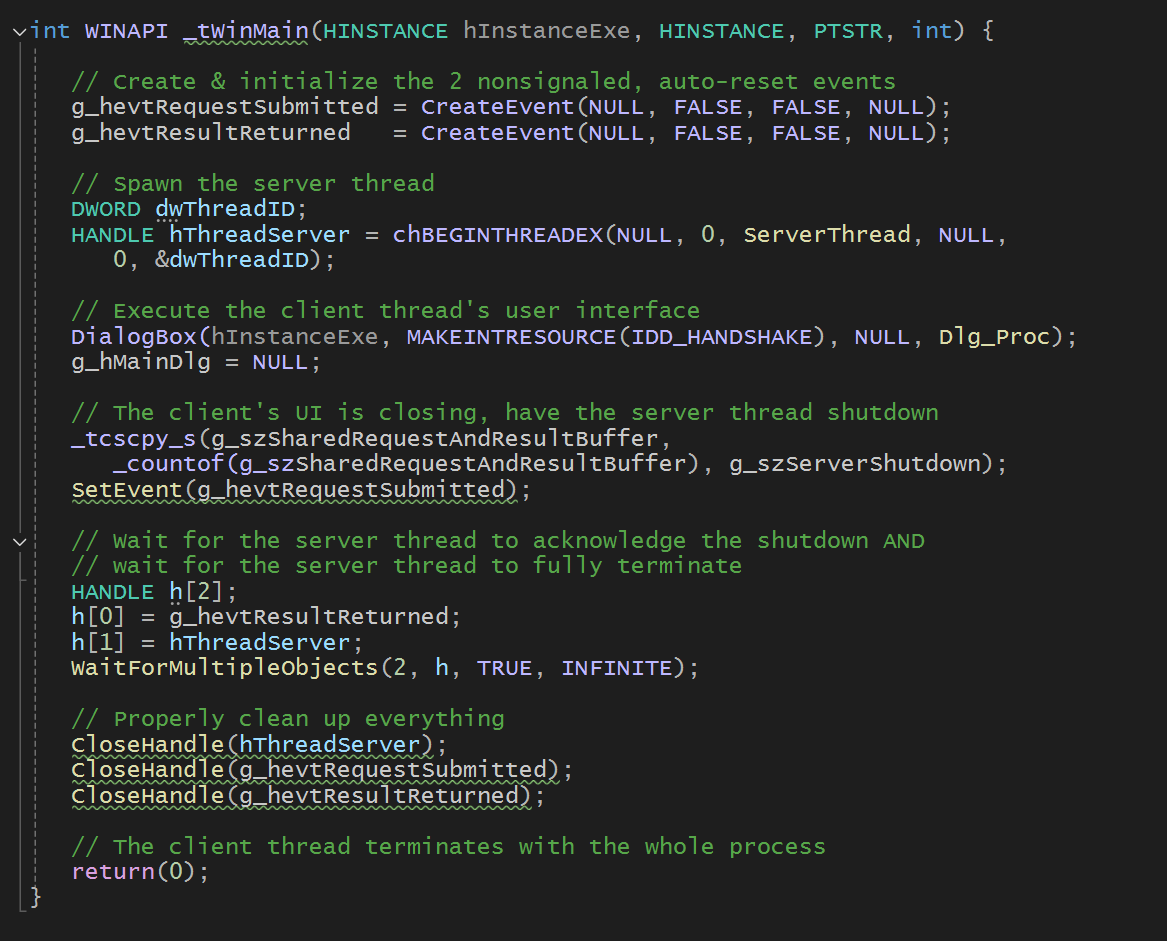
그리고 제가 위 함수들에서 Sleep을 호출하여 일정 간격 마다 스레드가 접근하는 것을 확인하였습니다. 즉, 대기하던 스레드들이 수동 리셋 버전과는 다르게 한번에 공유 데이터에 접근하는 것이 아닌 배타적으로 접근하게 된다.

**(정리)**

자동 리셋 이벤트의 경우에는 대기하는 스레드들 중에 하나의 스레드만 깨우는 것이고, 수동 리셋 이벤트는 대기하던 여러 스레드들이 스케줄 되는 것이다.

하지만 자동 리셋의 경우에는 해당 이벤트 객체를 논시그널로 시스템이 자동으로 변경해준다는 점이 있고, 수동 리셋의 경우에는 사용자가 직접 ResetEvent함수를 호출하여 해당 이벤트 객체를 논시그널 상태로 바꿔야하는 차이점이 있다.

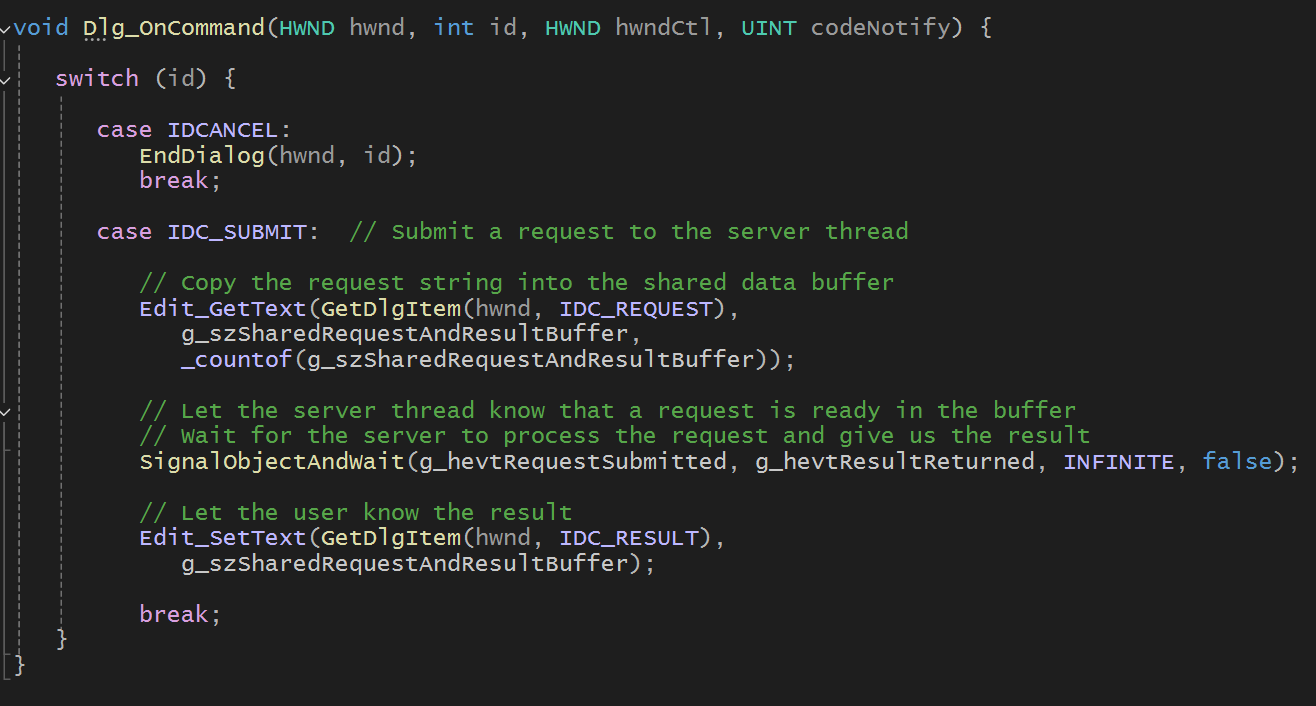
**(다운 받은 실습코드)**



클라이언트 스레드와 서버 스레드가 있다. **(실제로 클라와 서버가 연결한 것은 아님)**

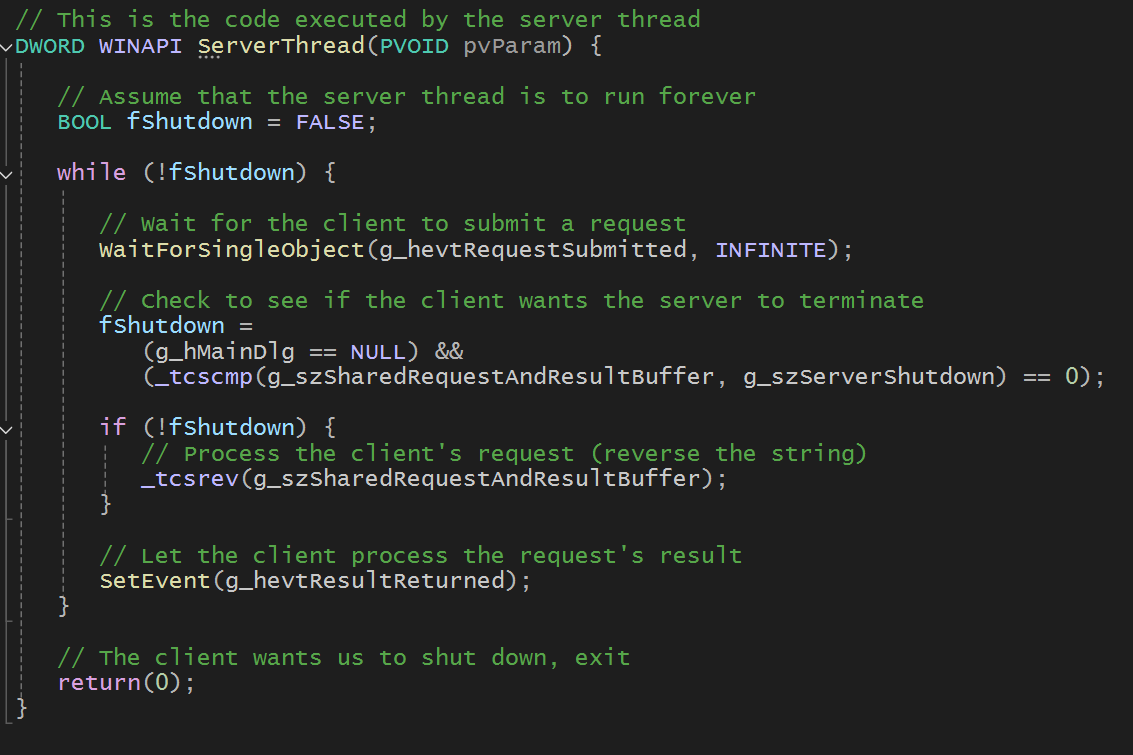
클라이언트 스레드에서는 대화 상자창에서 사용자의 입력을 받으면 서버에 event를 통해 signal을 줘서, 사용자의 입력을 받은 버퍼를 뒤집어서 다시 클라이언트에게 알려주는 것이다.

그래서 실습 코드에서는 2개의 이벤트 오브젝트(자동리셋)가 있으며, 클라와 서버가 공용으로접근할 사용자의 입력을 받는 버퍼가 전역으로 있다.



위는 클라이언트 스레드가 실행하는 main함수이다. 대화상자에서 사용자의 입력을 공용버퍼에 쓰고 SignalObjectAndWait 함수를 통해서 서버 스레드에게 통지하여 깨우는 코드이다.

참고로 SignalObjectAndWait 함수는 첫번째 매개변수를 시그널 상태로 만들고, 두번째 매개변수가 Signal 상태가 될 때까지 대기하는 함수이다. 즉, 방금 클라이언트 스레드에서 사용자의 입력을 버퍼에 저장해서 서버 스레드를 깨웠고, 서버스레드에서 작업을 마치고 SignalObjectAndWait의 두번째 매개변수를 Signal상태로 만들어주면 클라이언트 스레드는 다시 스케줄 가능 상태가 되는 것이다.



위 코드는 서버 스레드가 main으로 돌리는 함수이다. 아까 클라이언트 스레드에서 사용자의 입력을 받는다고 하였는데, 그 전까지는 대기를 하고있는 것이다. 그리고 사용자의 입력이 공용 버퍼로 들어오고 클라이언트에서 신호를 주면 스케줄이 되어서 공용버퍼에 있는 문자열을 뒤집고, 다시 클라이언트 스레드에게 통지를 하는 것이다.

위 실습코드는 이벤트 오브젝트를 이용하여 서로 다르 스레드끼리 데이터를 다루고 주고 받는 코드이다.

**세마포어 커널 오브젝트**

세마포어 커널 오브젝트는 리소스의 개수를 고려해야하는 상황에서 주로 사용된다.

세마포어는 다른 커널 오브젝트와 마찬가지로 사용 카운트를 가지고 있으며, 이 외에도 2개의 32비트 값을 가지고 있어서 최대 리소스 카운트와 현재 리소스 카운트를 저장하고 있다.

최대 리소스 : 세마포어가 제어할 수 있는 리소스의 최대 개수

현재 리소스 : 사용 가능한 리소스의 개수

**<스레드 풀과 세마포어를 활용한 예시 >**

클라이언트의 요청이 들어오면 서버를 이를 처리하는 상황이다. 서버에서 스레드 풀을 생성하여 최대 5개의 스레드만 준비를 한다. 세마포어를 활용한다면 앞서 말한 시나리오를 매우 훌륭한 방법으로 리소스를 관리하고 스레드의 스케줄링 상태를 제어할 수 있다.

세마포어의 최대 리소르 카운트를 5로 설정하고, 현재 리소스 카운트를 0으로 설정하여 어떤 클라이언트도 서비스를 요청하지 않은 상태로 초기화한다.

클라이언트의 요청이 들어오면 현재 리소스 카운트를 증가시키고, 요청을 완료하여 서버의 스레드 풀에 스레드가 되돌아오면 현재 리소스 카운트를 감소시킨다.

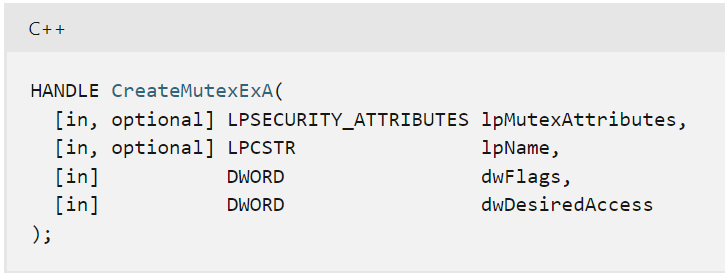
* 현재 리소스 카운트가 0보다 크면 세마포어는 시그널 상태가 된다
* 현재 리소스 카운트가 0이면 세마포어는 논시그널 상태가 된다
* 시스템은 현재 리소스 카운트를 음수로 만들 수 없다.
* 현재 리소스 카운트는 최대 리소스 카운트보다 커질 수 없다.

**뮤텍스 커널 오브젝트**

뮤텍스 커널 오브젝트는 스레드가 단일의 리소스에 대해 배타적으로 접근할 수 있도록 해 준다.

크리티컬 섹션과 비슷하지만 커널 모드 객체이다. 커널 객체이니 느리지만 서로 다른 프로세스에서 접근이 가능하다. 그리고 리소스에 대한 접근 권한을 획득할 때 시간 제한을 지정할 수 있다는 장점이 있다. 뮤텍스가 가지고 있는 값중에 “**반복 카운터”** 라는 값이 있다.

반복 카운터 : 뮤텍스를 소유하고 있는 스레드가 몇 회나 반복적으로 뮤텍스를 소유하고자 했는지에 대한 횟수를 나타내는 값이다. 다수의 스레드가 공유 메모리를 접근하는 것을 막고 배타적으로 접근하는 것을 보장할 수 있기 때문에 공유 데이터의 무결성을 유지할 수 있다.



매개변수 dwFlags로 0을 전달하면 뮤텍스의 스레드ID와 반복 카운터는 0으로 설정되어서, 뮤텍스가 어떠한 스레드에 의해서도 소유되지 않았으며, 시그널 상태임을 나타낸다.

TRUE를 전달하면 뮤텍스를 생성한 스레드가 소유하게되며 뮤텍스는 논시그널 상태가 된다.

스레드가 뮤텍스에 대한 핸들을 이용하여 대기 함수를 호출하면, 내부에서는 뮤텍스의 스레드ID 값을 확인하여 0이면 대기함수를 호출한 스레드의 ID로 값이 설정되고 반복카운터는 1로 설정한 후 스레드를 스케줄 가능 상태로 만든다.

만일 뮤텍스의 스레드ID가 0이 아니라면 스레드는 대기 상태로 계속 남는다. 시스템은 이러한 스레드를 기억하고 있다가 뮤텍스의 스레드ID가 0으로 변경되면 즉시 대기하던 스레드의 ID로 바꾸고 반복 카운터를 1로 설정한 후 스레드를 스케줄 가능상태로 만든다. 뮤텍스 커널오브젝트에 대한 상태 확인과 변경 작업은 항상 **원자적**으로 이루어진다.

**뮤텍스 “버림 문제 (Abandonment issues)”**

뮤텍스는 다른 모든 커널 오브젝트와는 다르게 “스레드 소유권”의 개념을 가지고 있다.

스레드가 ReleaseMutext를 호출하여 뮤텍스의 소유권을 반환할 때, 내부적으로는 반환하려는 스레드 ID와 뮤텍스를 소유하고 있는 스레드의ID가 동일한지 판단 후, 동일하면 반복 카운트 값을 감소한다.

그런데 만약 스레드가 뮤텍스를 제대로 반환하지 않고 종료될 수 있는데 이를 “버림 문제” 라고 표현한다.

시스템은 뮤텍스와 스레드 커널 오브젝트를 계속해서 추적하고 있기 때문에 언제 뮤텍스가 버려졌는지 정확히 알 수 있다. 뮤텍스의 버림이 발생하면 버려진 뮤텍스의 스레드ID와 반복 카운터를 0으로 변경한다. 그리고 대기 중인 스레드가 있다면 “공평하게” 하나의 스레드를 선택한 후 뮤텍스의 소유권을 주고 해당 스레드를 스케줄 가능 상태로 만든다.

그러면 이 경우에는 해당 스레드는 반환 값으로 **“WAIT\_OBJECT\_0”**가 아닌 **“WAIT\_ABANDONED”** 라는 특별한 값을 반환 받는다.

--------------------------------------------------------------------------------------------------------------

진행 상황 유튜브 링크:

|  |  |  |  |
| --- | --- | --- | --- |
| **문제점 정리** |  | | |
| **해결방안** |  | | |
| **다음주차** | 8주차 | **다음기간** | 8. 12 ~ 8. 18 |
| **다음주 할일** | 8주차에 계획되어있는 내용들 | | |
| **지도 교수**  **Comment** |  | | |